Contents

Chapter 1

Introduction

1.1 Design Methodolgy

There are two design methodolgies.

Embedded buncha things.

Integrated Circuits the sexy stuff that we would love talk about them.

There three main type of ICs.

Programmable Logic Devic buncha AND and OR.

Field Programmable Gate Arrays buncha LUT. they are relatively cheap and easy to produce. programmed easily, usually done with USB - can't get easier! can it?!.

Application Specific Integrate Circuite customized but non-programmable. Longer design time and higher cost but high preformance and lower power consumptions. Good for big boys that are done playing with their FPGA. but first they gotta send their design (*Fabrication patten*) to the factory (*FAB*).

Now you may ask why should I use programmable logic. Firstly, because it is cool and eviroment friendly. DON'T be a climate change denier baster please :)). Here are more reason

1. Reduce time to mark (TTM).
2. used for prototyping.
3. Reconfigurable computing
4. Custom computing
5. Reusability for different designs (Cant get greener than this :p).

1.2 Hardware Description Languages

describes hardware, it is in the name -_-!

1. Popular HDLs (IEEE standard)

   (a) Verilog (we gonna use this mostly because it's the cooler kid)
   (b) VHDL (used for modeling mostly and thus not all constructs are synthesizable)

2. other HDLs (not standard! not good! so no need to know baby)

HDLs are concurrent and not sequential - just like the real world. It also has timing that is you can have clocks for sequential circuits. furhtermore, it supports desing hierarchy (donnu what this is).

$$\text{Logic Synthesis} = \text{Translation} \rightarrow \text{Optimization} \rightarrow \text{Mapping}$$

We use ASM, *Algorithm State Machine* for large scale integration. and we use CAD/IDE (HDLs) for very large scale integrattion.

Definition (Netlist): HDL describing logic gates.

1.3 Design flow

1. Design specification- logical and physical.
2. Behavioral description- for the circuit.
3. RLT description via HDLs.
4. Functional verification and testing. if you suck go back to item 3.
5. Logic Synthesis.
6. Gate-level netlist. produced by logic synthesis.
7. Logical verification and testing. use the gate list and run test. if it sucks go back to item 3.
8. Floor planning, automatic place & route. usually done automaticly in FPGA and manually in ASIC. this step is called back-end design or physical design.
9. Physical layout - done with CAD and sent to FAB. GDStool
10. Layout verification. before you send your stupid design check it. done by barghis and not us.
11. Implementation. enjoy your shitty devic now.

Chapter 2

ASM Chart

The **algorithmic state machine** is a method for designing finite state machines which describe the sequential operations of a digital system. Its a behavioral model using flowcharts suitable for LSI.

There are three main elements to a ASM chart. State box, conditional box and a decision box.

in state and conditional boxes, the commands takes one clock cycle to be executed. conditional box is combinational. decision box contains binary expressions. **ASM block** is composed of one state box and all the decision and conditional boxes connected to the exit path of the state box. It represents what happens in the system during one clock cycle.

Chapter 3

Verilog

Behavioral design describes circuit as an algorithm and structural design describes explicit circuit elements. A behavioral example for a full adder.

```
module adder(a,b,cin,s,cout);
    output s,cout;
    input a,b,cin;
    reg s, cout;
    always @(a or b or cin)
    begin
        s = a^b^cin;
        cout = a&b | a& cin | b& cin;
    end
endmodule
```

And an example for a structural design of the same full adder.

```
module adder(a,b,cin,s,cout)
    output s,cout;
    input a,b,cin;

    wire w1,w2,w3,w4,w5;
    xor g1(w1,a,b);
    xor g2(s,w1,cin);
    and g3(w2,a,b);
    and g4(w3,a,cin);
    and g5(w4,b,cin);
    or g6(w5,w2,w3);
    or g7(cout,w4,w5);
endmodule
```

Number are specified by:

- Radix ('b,'h,'o,'d)
- Bit-length
- Value

for example

- 4'b1011
- 12'habc
- 16'd255

3.0.1 4-value logic

The set of value are $\{0, 1, x, z\}$. x is for unknown and z is for high impedence. Signal strengths are (in order):

1. supply (Driving)
2. strong (Driving)
3. pull (Driving)
4. large (Storage)
5. weak (Driving)
6. medium (Storage)
7. small (Storage)
8. highz (High impedence)

wire is used to represent connections between hardware elements (default = z). reg for hardware as well but it retains its value until next assignment (default = x).

1. Syntax

```
    wire/reg [msb_index : lsb_index] data_id;
```

2. Example

```
        wire a;
        wire [7:0] bus;
        wire [31:0] busA, busB, busC;
        reg clock;
        reg [0:40] virtual_addr;

```

modules are models of hardware. internals not visible to enviroment. internals can be changed as long as the interface (ports) is not changed.

```
    module fulladd4 (sum, c_out,a,b,c_in)
    ...
    endmodule
```

Ports are the terminals (pins) which have three types (in, out, inout). For port declaration

```
    input a,b,sel;
    input signed [15:0] a,b;
    output signed [31:9] result;
    output reg signed [32:1] sun;
    inout [15:12] addr;
```

In a module, inputs are always of type net but output can be reg as well. inout can be of type net only. We can implement delays as such

```
    and #(delay-time) a1 (out,i1,i2);
    and #(rise-val,fall-val) a1 (out,i1,i2);
    and #(min:typ:max,min:typ:max) a1 (out,i1,i2);
    bufif0 #(rise-val,fall-val,turnoff-val) a1 (out,in,control);
```